Available online at www.sciencedirect.com
SCIENCE DIRECT®
Journal of Computational and Applied Mathematics 196 (2006) 115–131
JOURNAL OF COMPUTATIONAL AND APPLIED MATHEMATICS
www.elsevier.com/locate/cam

A new adaptive mesh refinement strategy for numerically solving evolutionary PDE's

Denise Burgarelli[a,*], Mauricio Kischinhevsky[b], Rodney Josué Biezuner[a]

[a] *Departamento de Matemática, ICEx, Universidade Federal de Minas Gerais, Av. Antônio Carlos 6627, Pampulha, CP 702, CEP 30123-970, Belo Horizonte, MG, Brazil*
[b] *Instituto de Computação, Universidade Federal Fluminense R. Passo da Pátria 156, Bloco E, 3o. andar, Niterói, RJ, 24210-240, Brazil*

Received 29 January 2004; received in revised form 11 August 2005

Abstract

A graph-based implementation of quadtree meshes for dealing with adaptive mesh refinement (AMR) in the numerical solution of evolutionary partial differential equations is discussed using finite volume methods. The technique displays a plug-in feature that allows replacement of a group of cells in any region of interest for another one with arbitrary refinement, and with only local changes occurring in the data structure. The data structure is also specially designed to minimize the number of operations needed in the AMR. Implementation of the new scheme allows flexibility in the levels of refinement of adjacent regions. Moreover, storage requirements and computational cost compare competitively with mesh refinement schemes based on hierarchical trees. Low storage is achieved for only the children nodes are stored when a refinement takes place. These nodes become part of a graph structure, thus motivating the denomination autonomous leaves graph (ALG) for the new scheme. Neighbors can then be reached without accessing their parent nodes. Additionally, linear-system solvers based on the minimization of functionals can be easily employed. ALG was not conceived with any particular problem or geometry in mind and can thus be applied to the study of several phenomena. Some test problems are used to illustrate the effectiveness of the technique.

Keywords: Adaptive mesh refinement; Space filling curve; Numerical simulation; PDE

1. Introduction

Numerical solution of PDEs may require the use of a mesh refinement strategy that concentrates more mesh points where the solution and/or its derivatives change rapidly. For time-dependent problems, adaptive methods become particularly important since, by the dynamic nature of such problems, migration (or occurrence) of regions of rapid solution change may happen.

Typically, there are two ways for modifying grids in time. The first one is to move the grid with the fronts (see, e.g., [4,11,10,16]). In the second class of methods the grid does not move in time, and the refinement is localized only where needed. In this case, at each time step the refined grid may have to be adjusted to reflect the problem dynamics (see, e.g., [2,3,5,6,12,17–21]). Besides, some authors prefer a mixed strategy of moving meshes and local mesh refinement (see, e.g., [1]).

* Corresponding author. Tel.: +55 31 3499 5793; fax: +55 31 3499 5797.
E-mail addresses: burgarel@mat.ufmg.br (D. Burgarelli), kisch@ic.uff.br (M. Kischinhevsky), rodney@mat.ufmg.br (R.J. Biezuner).

0377-0427/$ - see front matter © 2005 Elsevier B.V. All rights reserved.
doi:10.1016/j.cam.2005.08.013

A mesh is formed by cells that cover the entire domain where the solution of the partial differential equation (or system) is sought, each of them being associated to a node of the data structure. From a given initial mesh, represented by the root of a tree, children nodes with increased degree of refinement are created. Each node may now be a parent node for subsequent refinement. This procedure can be carried out in the regions where it is required by some refinement criterion according to the specific problem. In this way, submeshes with different degrees of refinement may be found in adjacent areas of the entire domain.

According to [23] this procedure can be regarded as a *quadtree*, a class of hierarchical data structures whose common property is that they are based on the principle of recursive decomposition of space (including binary trees, oct-trees or any other 2^d-trees).

In [14], a new data structure, the *fully threaded tree* (FTT), is presented and applied to fluid dynamics simulations. FTT represents an improvement when compared to the traditional tree-based approaches mentioned above. Each node (leaves or not) is provided with easy access to its children, neighbors and parents, by means of specially designed pointers. Moreover, in FTT access among neighboring cells is faster, because full tree traversals are replaced by communication through neighbors' parent nodes.

The objective of this paper is to present a new, simple, yet flexible and powerful, strategy within the group of methods that deal with the refinement at fixed grids, which provides local mesh refinement at low computational cost. In this paper, the tree-based implementation is replaced by a one-level-at-a-time approach, which yields a graph-like implementation in which the children nodes (i.e., leaves) become autonomous as their parent node is deleted. A graph is formed which connects nearest-neighboring children cells through direct links. Moreover, neighboring cells which were generated from different parent nodes can be linked directly, if they have the same refinement level, or indirectly through a transition node, in case they have (arbitrarily) different refinement levels. This scheme will be called the autonomous leaves graph (ALG).

Tree implementations of quadtrees permit that adjacent regions of the domain have widely different levels of refinement. As a consequence, in order to reach neighboring cells, the algorithm may require tree-traversals. FTT manages to overcome these costs by means of allowing only one level of refinement difference between neighbors. However, this constraint imposes a medium-range effect in order to adjust the mesh; namely, if a cell needs further refinement, its neighbors will have to be refined as well. There is a loss in the locality of refinement and a layer-like picture is obtained.

ALG manages to preserve locality as well as low computational costs. Locality is supported because of the graph-like structure, since each node has pointers either to another node or to a transition node. Neighboring cells are direct and quickly linked by the transition nodes, even when their refinement stages differ by several levels. The communication among neighbors is along these short paths. The corresponding processing time does not depend on the global number of cells (that is, $\mathcal{O}(1)$).

The ordering of the cells in the mesh is performed through a space filling curve. The advantage in using such a curve is that it provides a simple and systematic way to visit all cells in the mesh. The modified space filling Hilbert curve used here for the mesh ordering was proposed in [9] and described in detail in [7]. Space filling curves have a high degree of locality; see, for example, [22] or [23].

In this work, a finite volume formulation is employed. The framework established by ALG applies to finite elements as well, in the context of h-adaptivity (see comments at the end of this paper). In our numerical tests, we apply interface flux homogenization, which is proper to a finite volume formulation, computing the flux across the interface of neighboring cells and comparing it to refinement and unrefinement quotas chosen by the user in order to decide if a cell is to be refined or if a bunch should be unrefined. Other error estimates appropriate to a finite volume formulation, adapted to the particular problem being studied, can be used (see [15] and the references therein).

In Section 2, the data structure is presented in detail, including a description of the refinement and unrefinement process. Moreover, the algorithm for generating the modified Hilbert's curve (MHC) used in the total ordering of the set of mesh cells is exhibited. In Section 3, some numerical tests conducted for a parabolic equation and a hyperbolic equation are presented.

2. Data structure

The graph structure underlying ALG contains two types of linked nodes: *cell nodes*, which correspond to mesh cells, and *transition nodes*, which will be used to connect cell nodes of different refinement levels. In order to simplify the

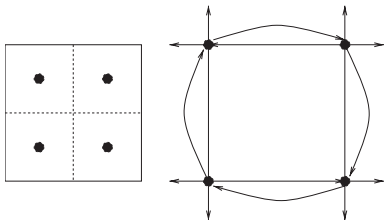

Fig. 1. Unit square and links for graph structure.

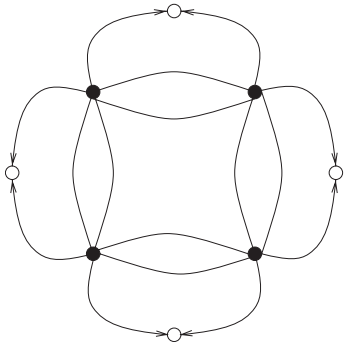

Fig. 2. Defining transition nodes.

presentation, a bidimensional mesh will be considered. In a bidimensional implementation, each cell node has four pointers, called *east*, *west*, *north* and *south*, oriented along corresponding namesake directions; these pointers can point either to neighboring cell nodes or to transition nodes. Each transition node has only three pointers, to connect cell nodes and/or transition nodes. Cell nodes have additional variables corresponding to the spatial coordinates of their centers and their physical states, whose number varies according to the problem being considered. Both nodes contain information regarding their *type* (cell or transition node) and their *level* of refinement.

2.1. Building blocks of the data structure

For ease of presentation consider the unit square, although all ideas can be immediately extended to arbitrary polygonal regions in two or more space dimensions.

Start with an initial mesh consisting of four cells inside the square, each of them identified by its center: $(\frac{1}{4}, \frac{1}{4})$, $(\frac{1}{4}, \frac{3}{4})$, $(\frac{3}{4}, \frac{3}{4})$ and $(\frac{3}{4}, \frac{1}{4})$, as shown in the left-hand side of Fig. 1. Each of the four cell nodes in Fig. 1 have links oriented along the four directions: *north*, along positive y direction; *south*, along negative y direction; *east*, along positive x direction; and *west*, along negative x direction.

As a result, we have the scheme presented on the right-hand side of Fig. 1. The remaining links that do not point to one of the four nodes in the square are then directed to the four transition nodes, displayed in Fig. 2 as white circles. Finally, as shown in Fig. 3, these four transition nodes are "grounded", i.e., they are linked to common ground.

In order to simplify the graphical presentation, the two arrows connecting two nodes will be replaced by one single line, as depicted in Fig. 4. Such a graph may be used to represent the basic links for an arbitrary square cell, since four lines (externally grounded at the initial stage) depart/arrive from the graph.

Four nodes with the same level of refinement which originated from the same node v are said to form a *bunch*, with parent v. The parent node is deleted, but each node is left with an identifier to determine that the four cells belong to the same bunch (the identifiers will be necessary in the unrefinement process; see Section 2.3).

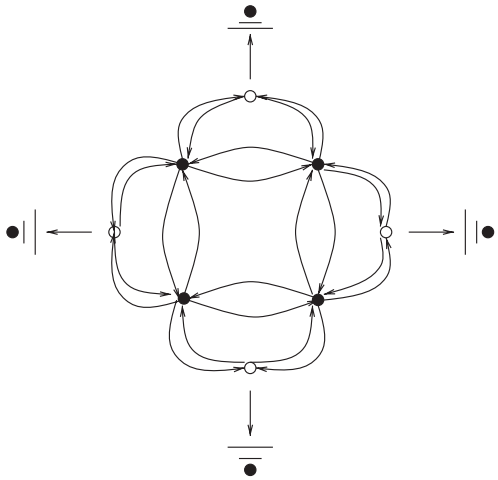

Fig. 3. Full scheme.

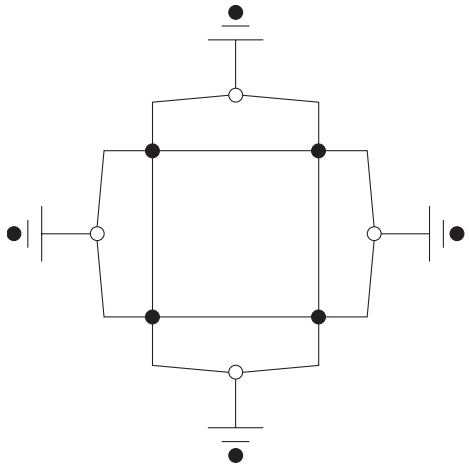

Fig. 4. Non-directional scheme.

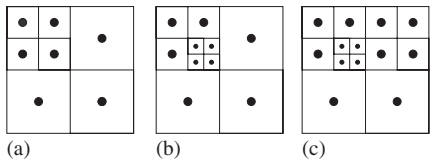

Fig. 5. A sequence of refinements.

2.2. *Mesh refinement*

In order to see how a refinement is performed, suppose that the cell centered at $(\frac{1}{4}, \frac{3}{4})$ is assigned for refinement by some criterion (determined by the problem; see Section 3 for some examples). In this case, a cell configuration as shown in Fig. 5(a) will be created.

This refinement is implemented by replacing the basic structure representing the links of the node to be refined (left frame in Fig. 6) by the structure developed for the unit square (right frame in Fig. 6). Each node (a, b) at the center of

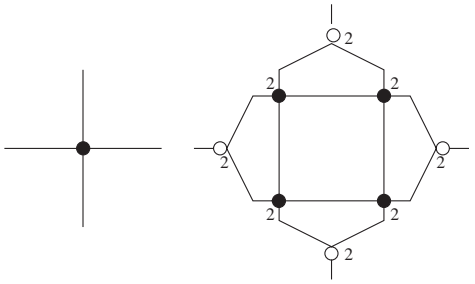

Fig. 6. Single cell's graph and elementary graph for refinement.

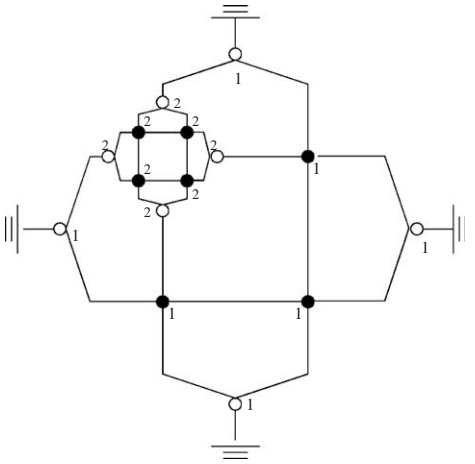

Fig. 7. Refining the northwest cell.

a cell with size c to be refined is replaced by the four nodes $(a - c/4, b - c/4)$, $(a - c/4, b + c/4)$, $(a + c/4, b + c/4)$ and $(a + c/4, b - c/4)$.

The four outward links departing from the transition nodes are connected to the nodes to which the four links of the node being replaced were connected. The resulting graph for the unit square becomes that of Fig. 7; the number placed next to each node corresponds to its level of refinement.

Next, consider an additional refinement as the one indicated in Fig. 5(b). Application of the same principle of replacing the local basic unit (left frame in Fig. 6) by the basic refinement unit (right frame in Fig. 6) leads to the structure depicted in Fig. 8.

Going one refinement step further, the configuration in Fig. 5(c) corresponds to the graph of Fig. 9. Note that, in this case, two adjacent transition nodes appear with the same refinement level, namely 2. This latest graph is simplified by eliminating these two redundant transition nodes, thus leading to the graph depicted in Fig. 10.

The sequence of refinements described above illustrates all processes during the grid refinement stage of the algorithm. Whenever two neighboring transition nodes at the same level are encountered, they are both deleted, simplifying the graph and ensuring that the search algorithm for neighbors of a cell will work efficiently. The search algorithm will find either the immediate neighbor or the transition nodes that will connect it with neighbors having different refinement levels.

Note that memory allocation requirements are very low, since it is needed only for nodes created during refinement. Moreover, the updates that occur during refinement, including simplification of graphs, are very efficient ($\mathcal{O}(1)$).

2.3. Mesh unrefinement

Before going into the details of the unrefinement procedure, it should be pointed out that, in order to preserve the basic building block structure of the graph, it is required that all four cell nodes to be replaced by a cell node with lower

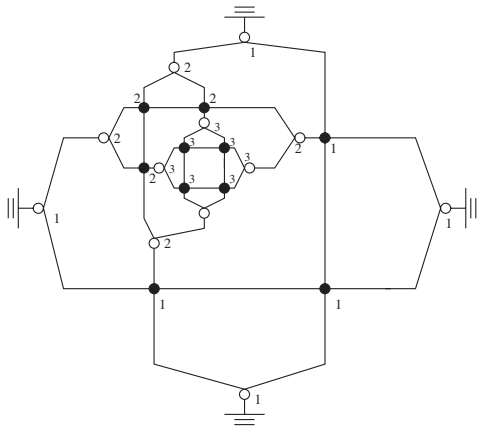

Fig. 8. One more level of refinement.

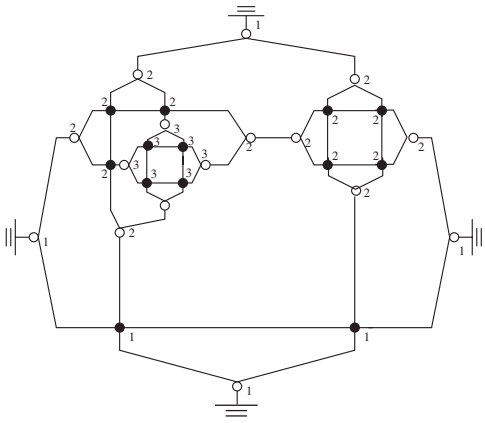

Fig. 9. Refining the northeast cell.

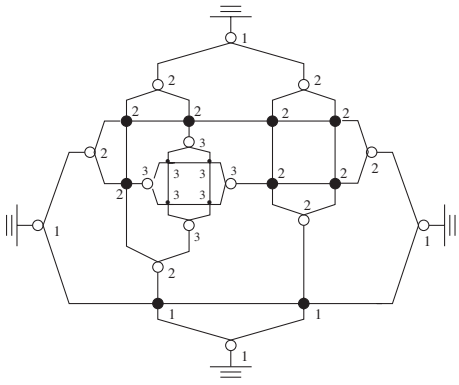

Fig. 10. Graph simplification.

refinement level must belong to the same bunch, allowing restoration of the previous configuration. This condition guarantees that previous configurations of the mesh can always be recovered performing mutually-independent local changes, after an arbitrary number of refinements and unrefinements have taken place.

As an example of the unrefinement process, consider the configuration in Fig. 11.

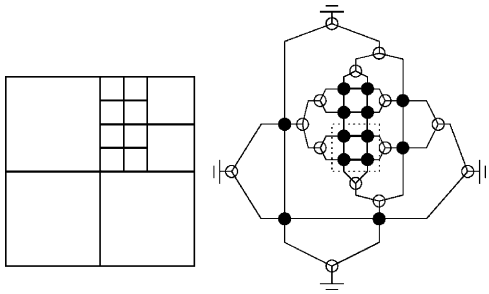

Fig. 11. Bunch selection.

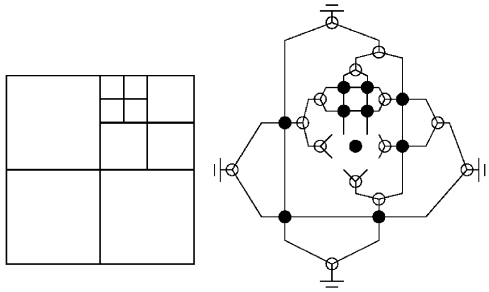

Fig. 12. Bunch collapse.

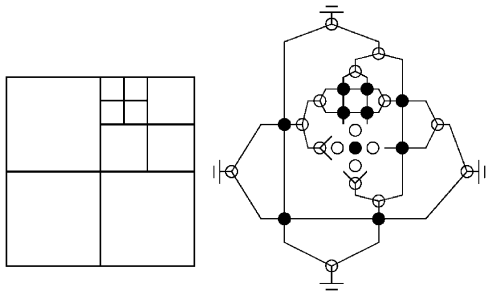

Fig. 13. Connecting nodes.

After the bunch marked in Fig. 11 has been replaced by a single node, one obtains the configuration shown in Fig. 12, in one unrefinement step.

The necessary stages for unrefining a bunch are as follows:

1. Transformation of the northeast node of the bunch in the resulting unrefined node (this node will be denoted by v_r).
2. Filling the resulting node with the new data (for instance, its space coordinates, level and physical constants related to the problem).
3. Connection of the resulting node with its neighbors.
4. Connection of the neighboring nodes with the resulting node.
5. Release of the memory space of the eliminated nodes.

The connection of the resulting node v_r with the neighboring nodes is done through the creation of four new transition nodes, each with the same level of refinement as the original bunch. This procedure is indicated in Fig. 13.

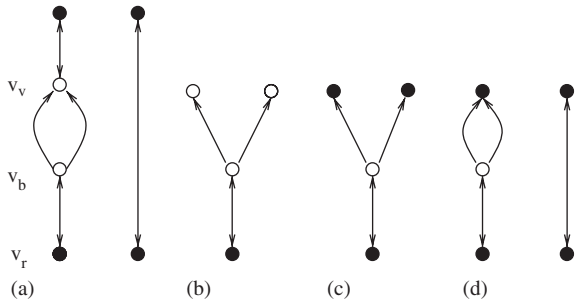

Fig. 14. Simplification by node elimination.

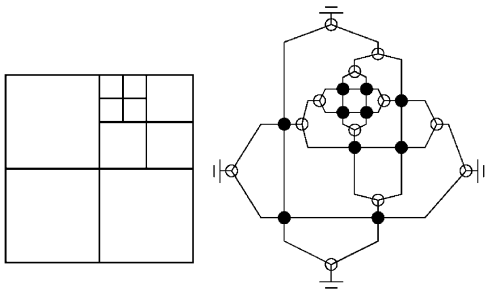

Fig. 15. Final configuration.

In the unrefinement process, one of the three pointers of each one of the four transition nodes is set to point to the resulting node v_r and the other two pointers are set to point to the neighboring nodes of the original bunch. For each configuration of neighboring nodes of v_r, a different strategy for connecting these two pointers is adopted. These strategies will be presently explained. In the following, v_b will denote one of the transition nodes created in the unrefinement process and v_v will denote a neighboring node to the original bunch, now adjacent to node v_b.

- If v_v is a transition node with the same level of refinement as v_b, after v_r is connected with the neighbor of v_v a simplification is made in order to eliminate v_v and v_b (Fig. 14(a)).
- If v_v is a transition node and nodes v_b and v_v have different levels of refinement, the created node v_b is kept, and it will make the connection of the resulting node v_r with the structure of the unrefined mesh (Fig. 14(b)).
- If v_v is not a transition node and nodes v_v and v_r have different levels of refinement, the created transition node v_b is preserved, and it makes the connection of v_r with v_v (Fig. 14(c)).
- If v_v is not a transition node and has the same level of refinement as v_r, the created transition node v_b is eliminated, and node v_r is connected directly to node v_v (Fig. 14(d)).

Adopting the strategies above, the configuration depicted in Fig. 11 becomes that of Fig. 15. Note the three simplifications that occurred in this example.

2.4. Total mesh ordering

In each of the cell nodes, an extra pair of pointers is present. These two pointers are used to create a total ordering of all cell nodes in the structure. Thus, starting from the first cell node, one can define a double linked list connecting all cell nodes. Every time a refinement is made, the four cell nodes generated are properly inserted in the linked list. Therefore, as each modification in the list is merely local, a new algorithm based on the construction of Hilbert's curve is used to implement the total ordering of the nodes. The construction of Hilbert's curve is described, for instance, in [24].

As an example, in the special case when the domain is uniformly partitioned, the points of the mesh are ordered as in Figs. 16–18, for the different degrees of refinement shown. If one follows the curve drawn by the edges that

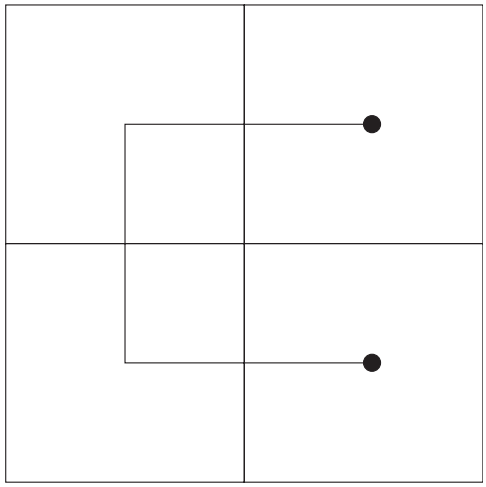

Fig. 16. First step.

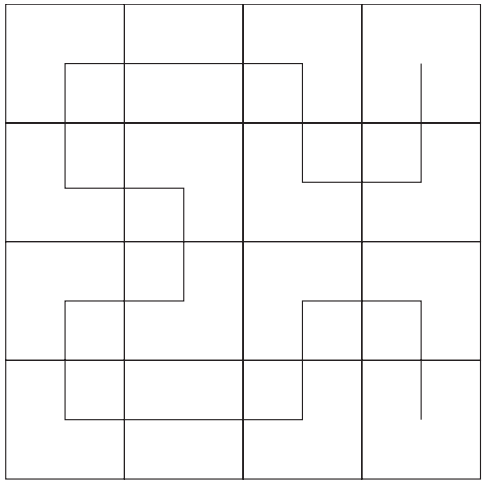

Fig. 17. Second step.

connect each cell node to its successor, we obtain a curve similar to the curves generated in the construction of Hilbert's curve.

When the domain is non-uniformly partitioned, the scheme above is modified as follows. During the uniform refinement employed at each stage above, all the cells were refined. In the modified algorithm developed in this work, called the MHC algorithm, refinements do not necessarily take place in every cell during the refinement stages. As a consequence, several levels of refinement may occur at the same stage, in different regions of the domain. The number of links per unit area, in each region, is directly related to the level of refinement of the corresponding cells connected by the curve within the same region. As an example, consider Fig. 19.

In order to construct the MHC, two numbers assigned to each cell node, n_0 and *level*, are needed. Number *level* denotes the level of refinement of the node, and n_0 is a number written in base 4 which describes the spatial position of the associated cell, as indicated by the MHC in that refinement stage. Initially, *level* is set to 1 for all four nodes, and the values of n_0 are as in Fig. 20 (each node level is indicated within parenthesis).

When a node is refined, its level is increased by one unit. The nodes just created need to be arranged and inserted in an orderly manner in the previous set of nodes. For instance, if the node with $n_0 = 1$ is refined, the scheme shown in Fig. 21 is obtained. The numbers 01, 11, 21 and 31 in base 4 indicate the position of the nodes just created,

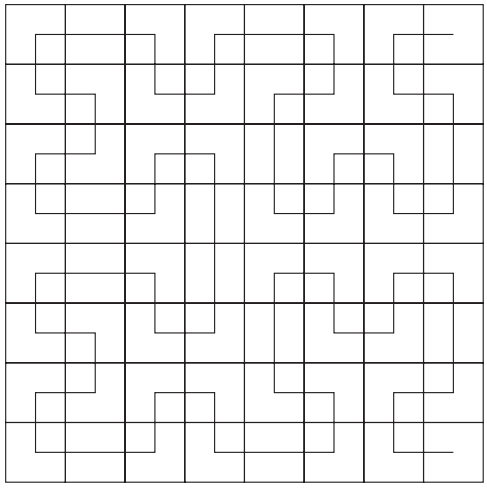

Fig. 18. Third step.

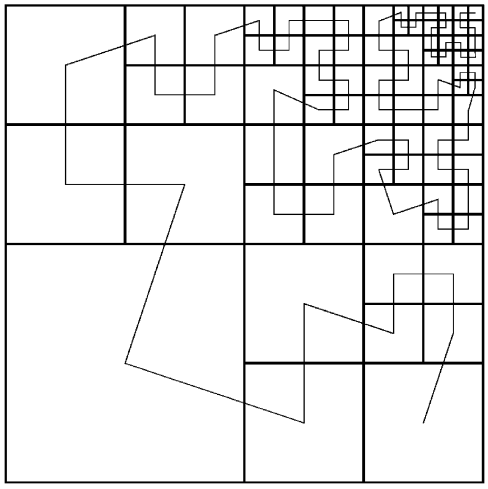

Fig. 19. Instance of a modified Hilbert's curve (MHC).

all of them with *level* = 2. The program then orders nodes 01, 11, 21 and 31. This ordering is done in the way described next.

There are four basic units that, conveniently joined, will form the MHC curve. They are shown in the diagrams below:

Starting from the original node position n_0, the newly created nodes will have positions n_0, $n_0 + p$, $n_0 + 2p$, $n_0 + 3p$, where $p = 4^{\text{level}}$. The original node was connected with two others nodes in the MHC, the *previous* and the *next* nodes. After refinement of the original node, the previous node connects with the node of position n_0, and the next node connects with the node of position $n_0 + 3p$. Now, it remains to show how the program decides which of the four basic units will be chosen each time a node is refined. Using the table

$$
\begin{array}{l}
\text{mhc}[0][0] = 1;\ \text{mhc}[1][0] = 0;\ \text{mhc}[2][0] = 3;\ \text{mhc}[3][0] = 2;\\
\text{mhc}[0][1] = 0;\ \text{mhc}[1][1] = 1;\ \text{mhc}[2][1] = 2;\ \text{mhc}[3][1] = 3;\\
\text{mhc}[0][2] = 0;\ \text{mhc}[1][2] = 1;\ \text{mhc}[2][2] = 2;\ \text{mhc}[3][2] = 3;\\
\text{mhc}[0][3] = 3;\ \text{mhc}[1][3] = 2;\ \text{mhc}[2][3] = 1;\ \text{mhc}[3][3] = 0;
\end{array}
$$

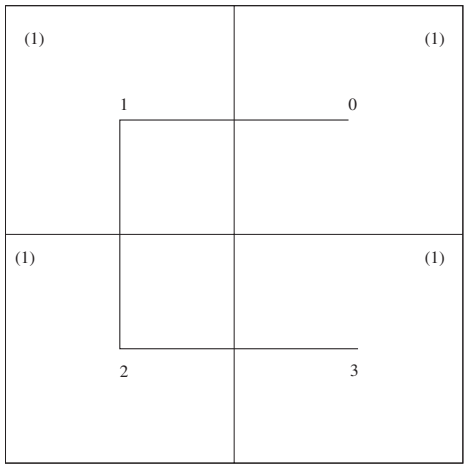

Fig. 20. First step (MHC).

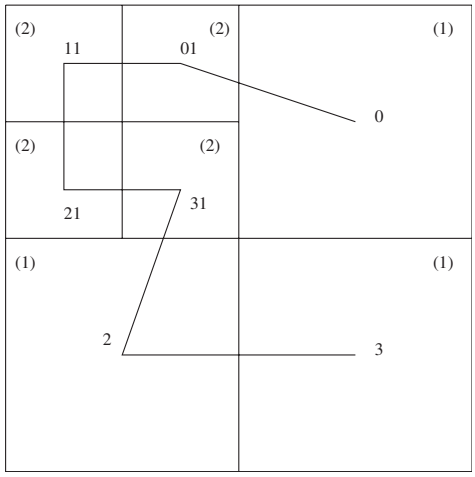

Fig. 21. A MHC with one refinement.

we execute the following algorithm

```
function buildMHC (n0, level)
{
  n1 = n0;
  i = 0;
  for (k = 1; k < level;  k++) {
    j = n1 mod 4;
    i = mhc[i][j];
    n1 = n1/4;
  }
  return i;
}
```

where the resulting number i is the chosen type of the basic unit for this new bunch, according to Fig. 22.

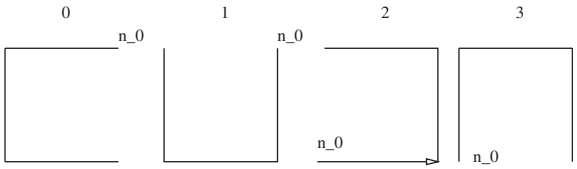

Fig. 22. The four basic units that form the MHC.

3. Numerical tests

In this section, we present some numerical experiments. The main purpose of the heat equation tests is to compare the time spent in refinement/unrefinement with time spent on actual numerical computations. In the wave equation test, the existence of cells whose neighbors have several different levels of refinement is illustrated. This feature of ALG, which shows up even in the context of smooth varying solutions, will be valuable in situations where the solution exhibits abrupt local variations. It should be emphasized that ALG was conceived as a general tool and thus applicable to a wide variety of problems. In [8], ALG was applied to the thermoacoustic problem, which involves solving a nonlinear system of Navier–Stokes equations. All tests here were carried on a WindowsXP/Pentium 4—2.8 GHz platform, using the GCC compiler.

3.1. Heat equation

Consider the two-dimensional heat conduction problem in a bounded region $\Omega \subset \mathbb{R}^2$, with continuous initial and boundary data:

$$\begin{aligned} &u_t = \nabla^2 u, \\ &u(\mathbf{x}, 0) = f(\mathbf{x}), \quad \mathbf{x} \in \Omega, \\ &u(\mathbf{x}, t) = g(\mathbf{x}), \quad \mathbf{x} \in \partial\Omega, \quad t \geqslant 0. \end{aligned} \tag{1}$$

The discrete formulation for (1), using backward differences in time and $\Delta x = \Delta y$, is

$$\frac{u_{i,j}^{n+1} - u_{i,j}^{n}}{\Delta t} = (\nabla^2 u)_{i,j}^{n+1}. \tag{2}$$

Integrating in the control volume, we rewrite (2) as

$$(\Delta x)^2 u_{i,j}^{n+1} - \Delta t \left(\int_{v_{i,j}} \nabla^2 u \, \mathrm{d}v_{i,j} \right)^{n+1} = (\Delta x)^2 u_{i,j}^{n}. \tag{3}$$

The value of u in the previous instant, $u_{i,j}^{n}$, is added to the right-hand side of the system $\mathbf{Ax} = \mathbf{b}$ associated to Eq. (1). Each element of matrix $\mathbf{A}$ is multiplied by Δt. And the coefficients of the terms $u_{i,j}^{n+1}$ are added to the diagonal of $\mathbf{A}$, that is, to each diagonal element of $\mathbf{A}$ the value $(\Delta x)^2$ is added. It should be pointed out that this reinforces the diagonal dominance of the principal matrix $\mathbf{A}$, leading to more efficiency in the use of the conjugate gradient method.

The following algorithm describes the steps used for the numerical resolution of heat equation (1):

begin
  initialize u by linear interpolation along the boundary
  initialize Δt
  while $(t < t\_final)$
    update **b**
    assembles_coefficient_matrix

Table 1
Numerical results for the heat equation

Boundary condition	L	MaxN	RefTime	UnrefTime	CompTime	(%)
	5	856	6.8×10^{-3}	9.5×10^{-3}	5.6×10^{-2}	28.9
$g(x, y) = 10$	6	3016	1.9×10^{-2}	7.9×10^{-2}	0.8	11.7
	7	3580	3.5×10^{-2}	7.7×10^{-2}	2.5	4.5
	8	34 852	2.7×10^{-1}	2.9	73.9	4.2
	5	1024	7.6×10^{-3}	4.2×10^{-3}	1.5×10^{-1}	7.9
$g(x, y) = x - y$	6	4090	5.0×10^{-2}	2.5×10^{-2}	3.2	2.3
	7	15682	1.2×10^{-1}	8.8×10^{-2}	21.6	1.0
	8	65 530	4.7×10^{-1}	3.3×10^{-1}	83.8	1.0
	5	934	7.5×10^{-3}	5.4×10^{-3}	0.2	6.5
$g(x, y) = x^2 - y^2$	6	3529	3.4×10^{-2}	5.9×10^{-2}	4.7	2.0
	7	12 442	1.2×10^{-1}	8.5×10^{-2}	28.6	0.7
	8	55 414	4.5×10^{-1}	8.3×10^{-1}	169.9	0.8

L = maximum refinement level reached, MaxN = maximum number of cells attained in the grid, RefTime = total time spent in refinement, UnrefTime = total time spent in unrefinement, CompTime = total time spent in numerical computations, % = 100 × (RefTime + UnrefTime)/CompTime.

```
        conjugate_gradient
        refine_derefine
        t = t + Δt
    end while
end
```

The criteria used for refinement and unrefinement are based on the flux across the interface of neighboring cells. That is, if the absolute value of the flux is larger than a refinement quota chosen by the user, the program chooses to refine this cell, while if the absolute value of the flux of all four cells of a bunch is less than an unrefinement quota chosen by the user, the program chooses to unrefine the bunch. The choice of the user is based on his requirements of a greater or lesser refinement in certain regions of the domain where there is greater or lesser variation in the behavior of the solution, respectively. These criteria overestimate the number of refinements needed, since a big difference of fluxes does not necessarily mean that there is a big variation in the difference between the approximate and exact solutions, for this variation can already occur in the exact solution. An error estimator, tailored to specific applications, can be used in order to improve the performance of the algorithm.

The tests were performed choosing $\Delta t = 0.1$ with the program running until $t = 10\,\Delta t$. In all problems, the boundary condition was chosen as the steady-state solution intended, and the initial condition was set as $f(\mathbf{x}, 0) \equiv 0$. At level L, the minimum cell size attained is 2^{-L}. The test results are summarized in Table 1 (times are given in seconds).

Table 1 shows that most of the computing time in ALG is spent in the core of the computation, leaving very low processing time to refine/unrefine stages. Moreover, the more high resolution of results is required, the more the computational cost in refine/unrefine stages decreases.

3.2. Wave equation

The two-dimensional first-order wave equation is

$$\frac{\partial u}{\partial t} + c_1 \frac{\partial u}{\partial x} + c_2 \frac{\partial u}{\partial y} = 0. \tag{4}$$

Consider two travelling waves moving in opposite directions, along a direction not coincident with the coordinated axes. In particular, consider $c_1 = c_2 = \frac{\sqrt{2}}{2}$ and $u = u_1(x, y, t) + u_2(x, y, t)$, where u_1 satisfies

$$\frac{\partial u_1}{\partial t} + c_1 \frac{\partial u_1}{\partial x} + c_2 \frac{\partial u_1}{\partial y} = 0, \tag{5}$$

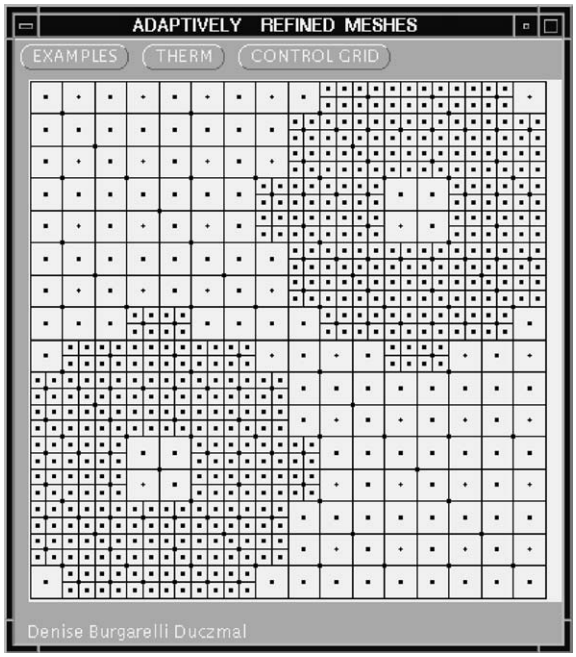

Fig. 23. Coarse mesh at $t = 0$.

with initial condition given by

$$u_1(x, y, 0) = \begin{cases} 1 - 16[(x - 0.25)^2 + (y - 0.25)^2] & \text{if } u_1(x, y, 0) \geqslant 0, \\ 0 & \text{if } u_1(x, y, 0) < 0 \end{cases} \tag{6}$$

and u_2 satisfies

$$\frac{\partial u_2}{\partial t} - c_1 \frac{\partial u_2}{\partial x} - c_2 \frac{\partial u_2}{\partial y} = 0, \tag{7}$$

with initial condition given by

$$u_2(x, y, 0) = \begin{cases} 1 - 16[(x - 0.75)^2 + (y - 0.75)^2] & \text{if } u_2(x, y, 0) \geqslant 0, \\ 0 & \text{if } u_2(x, y, 0) < 0. \end{cases} \tag{8}$$

The basic algorithm for numerically solving this problem, using the modified method of characteristics [13] is as follows:

begin
  initialize u with initial conditions
  initial refinement
  store u_old
  while ($t < t\_final$)
    method of characteristics
    store u_old
    refine_derefine
    $t = t + \Delta t$
  while end
end

The procedure to find the value in the cell that contains the coordinates of the foot of a characteristic line (i.e., the intersection point of a characteristic line with the computational domain) in the previous instant is now described.

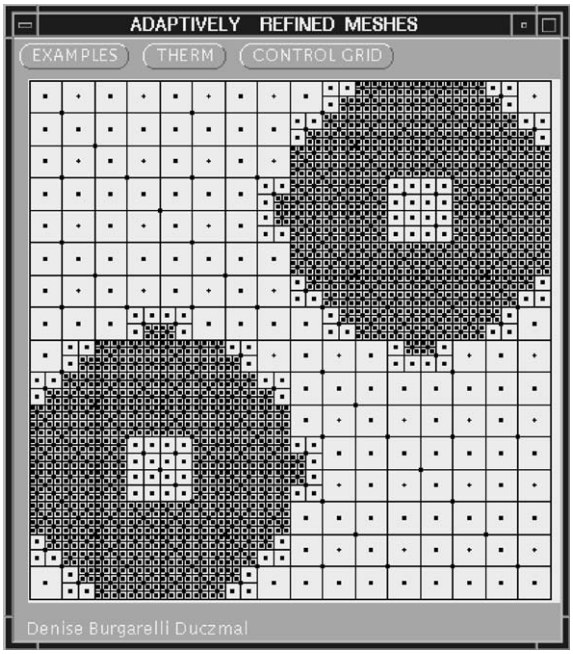

Fig. 24. Finer mesh at $t = 0$.

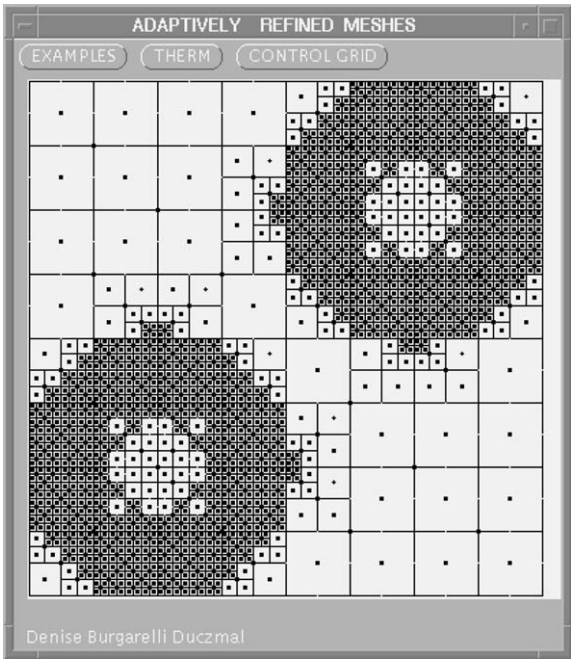

Fig. 25. Unrefinement made.

In the linear approach, if (x_c, y_c) denote the coordinates of the current cell, the coordinates (x_p, y_p) of the foot of the characteristic line that intersects this cell are obtained by

$$x_p = x_c - c_1 \Delta t,$$
$$y_p = y_c - c_2 \Delta t. \tag{9}$$

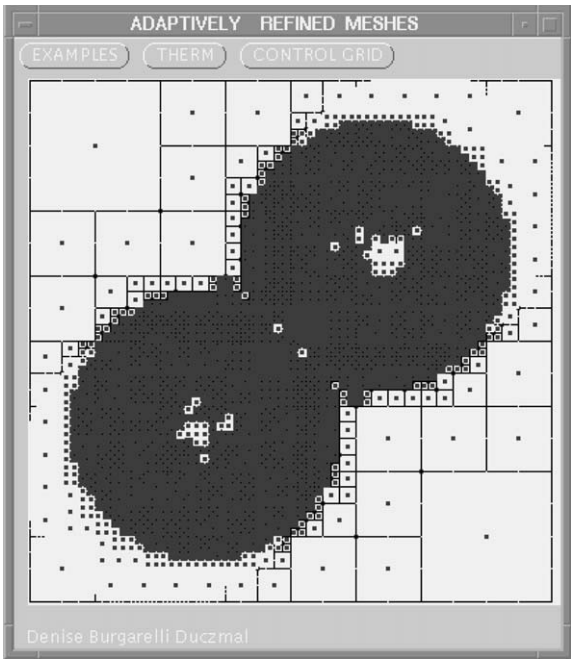

Fig. 26. Waves starts to overlap.

After calculating the coordinates of the foot of a characteristic line, one needs to find the cell that corresponds to these coordinates. The starting point is the cell which, going back in time, meets the characteristic line. The walking direction is chosen through comparison of the horizontal and vertical distances between the center of the current cell and the foot of the characteristic line. The direction of the largest distance is taken as the walking direction, moving thus to the contiguous cell, which then becomes the current cell for walking, and the process continues until the cell that contains the foot of the characteristic line is reached.

Figs. 23–26 illustrate this example. We begin with an uniform mesh with cell side length $\frac{1}{16}$. Fig. 23 shows the mesh refined for this problem at the instant $t = 0$, with cell side lengths no less than $\frac{1}{32}$. Fig. 24 shows the same pattern with more refinement (cell side lengths $\geqslant \frac{1}{64}$). In the following, the program sweeps the entire mesh trying to detect if it is possible to do some unrefinement that fulfills the required tolerance for the problem. Some areas are then unrefined (as for instance, where the solution has zero value). Fig. 26 displays a stage of refinement/unrefinement for a instant of time when the two waves overlap. Notice the coexistence of subregions with widely different degrees of refinement in Figs. 24–26.

4. Concluding remarks

ALG, a new technique for structured AMR, and its companion set of computational tools, was introduced. Among the advantages of this new procedure, besides locality, are the computational efficiency and the possibility of the coexistence of subregions with widely different degrees of refinement, as demonstrated by the numerical tests.

A notable feature of ALG, the fact that it allows the replacement of a group of cells by another with an arbitrary level of refinement, can be extended and easily applied to higher-dimensional problems, as well as to triangular meshes.

ALG can be used in several types of PDE problems and in different geometries, with adaptive refinement of complex boundaries.

This paper discusses in detail a finite volume formulation, but there is no reason to preclude the use of ALG in a finite element formulation in the context of h-adaptivity. However, several issues must be dealt with, such as the generation of the triangular mesh and its ordering.

Acknowledgements

We thank the reviewers and the editor for their valuable comments and suggestions.

References

[1] D.C. Arney, J.E. Flaherty, An adaptive mesh-moving and local refinement method for time-dependent partial differential equations, ACM Trans. Math. Software 16 (1990) 48–71.
[2] I. Babuska, J.P.S.R. Gago, D.W. Kelly, C. Zienkiewicz, A posteriori error analysis and adaptive processes in the finite element method: part II—Adaptive mesh refinement, Internat. J. Numer. Methods Eng. 19 (1982) 1621–1656.
[3] I. Babuska, W.C. Rheinboldt, Error estimates for adaptive finite element computations, SIAM J. Numer. Anal. 15 (4) (1978) 736–754.
[4] M.J. Baines, A.J. Wathen, Moving finite element modelling of compressible flow, Appl. Numer. Math. 2 (1986) 495–514.
[5] J. Bell, M. Berger, J. Saltzman, M. Welcome, Three-dimensional adaptive mesh refinement for hyperbolic conservation laws, SIAM J. Sci. Comput. 15 (1) (1994) 127–138.
[6] M. Berger, J. Oliger, Adaptive methods for hyperbolic partial differential equations, J. Comput. Phys. 53 (1984) 484–512.
[7] D. Burgarelli, Modelagem computacional e simulação numérica adaptativa de equações diferenciais parciais evolutivas aplicadas a um problema termoacústico, Tese de doutorado, PUC-Rio, Rio de Janeiro, Brasil, 1998 (in Portuguese).
[8] D. Burgarelli, M. Kischinhevsky, Efficient numerical simulation of a simplified thermoacoustic engine with new adaptive mesh refinement tools, Computational Methods in Engineering'99, XX CILAMCE, São Paulo, SP, 1999.
[9] D. Burgarelli, M. Kischinhevsky, P.J. Paes Leme, O.T. Silveira, Refinamento de malha adaptativa em paralelo, (C_3AD), Colloquia em Computação Científica de Alto Desempenho, Rio de Janeiro, RJ, 1995 (in Portuguese).
[10] N. Carlson, K. Miller, Design and application of a gradient-weighted moving finite element code, part II, in 2-D, SIAM J. Sci. Comput. 19 (3) (1994) 766–798.
[11] N. Carlson, K. Miller, Design and application of a gradient-weighted moving finite element code, part I, in 1-D, SIAM J. Sci. Comput. 19 (3) (1998) 728–765.
[12] P. De Oliveira, On the caracterization of finite differences "optimal" meshes, J. Comput. Appl. Math. 36 (1991) 137–148.
[13] J. Douglas Jr., T.F. Russell, Numerical methods for convection-dominated diffusion problems based on combining the method of characteristics with finite element or finite difference procedures, SIAM J. Numer. Anal. 19 (5) (1982) 871–885.
[14] A.M. Khokhlov, Fully threaded tree algorithms for adaptive refinement fluid dynamics simulations, J. Comput. Phys. 143 (1998) 519–543.
[15] M.A. Martins, R.M. Valle, L.S. Oliveira, D. Burgarelli, Error estimation and adaptivity for finite volume methods on unstructured triangular meshes: elliptic heat transfer problems, Numer. Heat Transfer B 42 (2002) 461–483.
[16] K. Miller, R. Miller, Moving finite elements, Part I, SIAM J. Numer. Anal. 18 (1981) 1019–1032.
[17] W.F. Mitchell, Adaptive refinement for arbitrary finite-element spaces with hierarchical bases, J. Comput. Appl. Math. 36 (1991) 65–78.
[18] J. Oliger, X. Zhu, Stability and error estimation for component adaptive grid methods, Appl. Numer. Math. 20 (1996) 407–426.
[19] W.C. Rheinboldt, On a theory of mesh-refinement processes, SIAM J. Numer. Anal. 17 (6) (1980) 766–778.
[20] M.C. Rivara, Mesh refinement processes based on the generalized bisection of simplices, SIAM J. Numer. Anal. 21 (3) (1984) 604–612.
[21] M.C. Rivara, Local modification of meshes for adaptive and/or multigrid finite-element methods, J. Comput. Appl. Math. 36 (1991) 79–89.
[22] H. Sagan, Space-Filling Curves, Springer, Berlin, 1994.
[23] H. Samet, The Design and Analysis of Spatial Data Structures, Addison-Wesley, Reading, MA, 1990.
[24] N. Wirth, Algorithms + Data Structures = Programs, Prentice-Hall, Englewood Cliffs, NJ, 1976.